

# Code library

Yifei Shan

2021 年 12 月 8 日

目录

1	基本算法	1			
1.1	位运算	1			
1.2	前缀和与差分	1			
1.2.1	前缀和	1			
1.2.2	差分	1			
1.3	大整数	1			
1.3.1	大数	1			
1.3.2	加法	3			
1.3.3	减法	3			
1.3.4	乘法	4			
1.3.5	除法	4			
1.4	区间和	4			
1.5	set 技巧	5			
2	图论	5			
2.1	链式前向星	5			
2.2	树的 dfs 序	5			
2.3	树的深度	6			
2.4	图的连通块划分	6			
2.5	欧拉路	6			
2.6	强连通分量	6			
2.7	点分治	6			
2.8	树的重心	7			
2.9	树的直径	7			
2.9.1	两次 dfs	7			
2.9.2	树形 dp	8			
2.10	拓扑排序	8			
2.11	dijkstra	9			
2.11.1	朴素算法	9			
2.11.2	堆优化	10			
2.12	SPFA	10			
2.12.1	求最短路	10			
2.12.2	判断负环	10			
2.12.3	差分约束	10			
2.13	Floyd	10			
2.13.1	求关系闭包 (递闭包)	10			
2.13.2	多源最短路	11			
2.14	LCA(最近公共祖先)	11			
2.15	二分图匹配 (染色法)	12			
2.16	二分图最大匹配 (匈牙利算法)	12			
2.17	Tarjan 求割边	13			
2.18	Tarjan 求割点	13			
2.19	网络流	13			
2.19.1	最大流	13			
2.19.2	最小割	14			
2.19.3	有向无环图最小路径覆盖	15			
3	数据结构	16			
3.1	并查集	16			
3.2	分块	18			
3.3	树状数组	18			
3.4	线段树	18			
3.5	bfs	20			
3.5.1	小猫爬山	20			
3.5.2	八皇后	20			
4	数论	20			
4.1	筛质数	20			
4.1.1	线性筛	20			
4.2	质因数分解	20			
4.3	最大质因数	21			
4.4	正因数集合	21			
4.5	逆元	21			
4.5.1	费马小定理	21			
4.5.2	线性逆推	21			
4.5.3	扩展欧几里得	21			
5	计算几何	21			
5.1	点初始模板	21			
5.2	一些基本运算	21			
5.3	点模板总结	22			
5.4	线	23			
6	字符串	23			
6.1	函数	23			
6.2	Manacher	24			
6.3	KMP	24			
7	背包	24			
7.1	01	24			
7.2	完全背包	24			
7.3	多重背包	24			
7.4	分组背包	25			

# 1 基本算法

## 1.1 位运算

```

1 (n >> k) & 1; //取n的第k位
2 n & ((1 << k) - 1) //取前k位
3 n ^ (1 << k); //k位取反
4 n | (1 << k); //第k位取1
5 n & ~(1 << k) //第k位取0

```

```

21 while(t--){
22 {
23     int x,y,c;
24     cin>>x>>y>>c;
25     insert(x,y,c);
26 }
27 for(int i=1;i<=n;i++) b[i]+=b[i-1];
28 for(int i=1;i<=n;i++) cout<<b[i]<<" ";
29 }

```

## 1.2 前缀和与差分

### 1.2.1 前缀和

```

1 #include <iostream>
2 using namespace std;
3
4 const int N = 100010;
5 int n, m;
6 int a[N], s[N];
7
8 int main()
9 {
10     scanf("%d%d", &n, &m);
11     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
12
13     for (int i = 1; i <= n; i++) s[i] = s[i - 1] + a[i]; // 前缀和的初始化
14
15     while (m--)
16     {
17         int l, r;
18         scanf("%d%d", &l, &r);
19         printf("%d\n", s[r] - s[l - 1]); // 区间和的计算
20     }
21
22     return 0;
23 }

```

### 1.2.2 差分

```

1 #include <iostream>
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int N=1e6+10;
6 int p[N],tmp[N];
7 long long ans;
8 int b[N],a[N];
9 void insert(int l,int r,int c)
10 {
11     b[l]+=c;
12     b[r+1]-=c;
13 }
14
15 int main()
16 {
17     int t,n;
18     cin>>n>>t;
19     for(int i=1;i<=n;i++) cin>>a[i];
20     for(int i=1;i<=n;i++) insert(i,i,a[i]);

```

## 1.3 大整数

### 1.3.1 大数

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 struct BigInteger{
5     //BASE为vector数组中一位中最大存储的数字，前面都是以
6     //10计算的
7     //WIDTH为宽度
8     static const int BASE = 100000000;
9     static const int WIDTH = 8;
10     vector<int> s;
11     //正数为1，负数为-1
12     int flag = 1;
13
14     //构造函数
15     BigInteger(ll num = 0){*this = num;}
16     BigInteger(string str){*this = str;}
17     BigInteger(const BigInteger& t){
18         this->flag = t.flag;
19         this->s = t.s;
20     }
21     //赋值函数
22     BigInteger operator = (ll num){
23         s.clear();
24         do {
25             s.push_back(num % BASE);
26             num /= BASE;
27         }while(num > 0);
28         return *this;
29     }
30     BigInteger operator = (string &str){
31         s.clear();
32         int x,len = (str.length()-1)/WIDTH + 1;
33         for(int i=0;i<len;i++){
34             int end = str.length() - i*WIDTH;
35             int start = max(0,end - WIDTH);
36             sscanf(str.substr(start,end-start).c_str(),
37                 "%d",&x);
38             s.push_back(x);
39         }
40         return *this;
41     }
42     //基本比较函数 A < B
43     bool cmp( vector<int> &A, vector<int> &B){
44         if(A.size() != B.size())return A.size() < B.size();
45         for(int i=A.size()-1;i>=0;i--){
46             if(A[i] != B[i]){
47                 return A[i] < B[i];

```

```

48     }
49 }
50 return false;
51 }
52 //比较函数如果小于则返回真
53 bool operator < ( BigInteger & b){
54     return cmp(s,b.s);
55 }
56 bool operator > ( BigInteger& b){
57     return b < *this;
58 }
59 bool operator <= ( BigInteger &b){
60     return !(b < *this);
61 }
62 bool operator >= ( BigInteger &b){
63     return !(*this < b);
64 }
65 bool operator == ( BigInteger &b){
66     return !(b < *this) && (*this < b);
67 }
68 //基本四则运算
69 vector<int> add(vector<int> &A, vector<int> &B);
70 vector<int> sub(vector<int> &A, vector<int> &B);
71 vector<int> mul(vector<int> &A, int b);
72 vector<int> mul(vector<int> &A, vector<int> &B);
73 vector<int> div(vector<int> &A, int b);
74 vector<int> div(vector<int> A, vector<int> B);
75
76 //重载运算符
77 BigInteger operator + (BigInteger &b);
78 BigInteger operator - (BigInteger &b);
79 BigInteger operator * (BigInteger &b);
80 BigInteger operator * (int& b);
81 BigInteger operator / (BigInteger & b);
82 BigInteger operator / (int b);
83 };
84 //重载<<
85 ostream& operator << (ostream &out,const BigInteger&
86     x) {
87     if(x.flag == -1)out << '-';
88     out << x.s.back();
89     for(int i = x.s.size() - 2; i >= 0;i--){
90         char buf[20];
91         sprintf(buf,"%08d",x.s[i]);//08d此处的8应该与
92             WIDTH一致
93         for(int j=0;j<strlen(buf);j++)out<<buf[j];
94     }
95     return out;
96 }
97 //重载输入
98 istream& operator >> (istream & in,BigInteger & x){
99     string s;
100     if(!(in>>s))return in;
101     x = s;
102     return in;
103 }
104 vector<int> BigInteger::add( vector<int> &A, vector<
105     int> &B){
106     if(A.size() < B.size())return add(B,A);
107     int t = 0;
108     vector<int> C;
109     for(int i=0;i<A.size();i++){
110         if(i<B.size())t += B[i];
111         t += A[i];
112         C.push_back(t%BASE);

```

```

110         t /= BASE;
111     }
112     if(t)C.push_back(t);
113     while(C.size() > 1 && C.back() == 0)C.pop_back();
114     return C;
115 }
116 vector<int> BigInteger::sub( vector<int> &A, vector<
117     int> &B){
118     vector<int> C;
119     for(int i=0,t=0;i<A.size();i++){
120         t = A[i] - t;
121         if(i<B.size())t -= B[i];
122         C.push_back((t+BASE)%BASE);
123         if(t < 0)t = 1;
124         else t = 0;
125     }
126     while(C.size() > 1 && C.back() == 0)C.pop_back();
127     return C;
128 }
129 vector<int> BigInteger::mul(vector<int> &A,int b){
130     vector<int> C;
131     int t = 0;
132     for(int i = 0;i < A.size() || t; i++){
133         if(i < A.size()) t += A[i] * b;
134         C.push_back(t%BASE);
135         t /= BASE;
136     }
137     return C;
138 }
139 //大数乘大数乘法需要将BASE设置为10, WIDTH设置为1
140 vector<int> BigInteger::mul( vector<int> &A, vector<
141     int> &B) {
142     int la = A.size(),lb = B.size();
143     vector<int> C(la+lb+10,0);
144     for(int i=0;i<la;i++){
145         for(int j=0;j<lb;j++){
146             C[i+j] += A[i] * B[j];
147         }
148     }
149     for(int i=0;i<C.size();i++){
150         if(C[i] >= BASE){
151             C[i + 1] += C[i] / BASE;
152             C[i] %= BASE;
153         }
154     }
155     while(C.size() > 1 && C.back() == 0)C.pop_back();
156     return C;
157 }
158 //大数除以整数
159 vector<int> BigInteger::div(vector<int> & A,int b){
160     vector<int> C;
161     int r = 0;
162     for(int i = A.size() - 1;i >= 0;i--){
163         r = r * 10 + A[i];
164         C.push_back(r/b);
165         r %= b;
166     }
167     reverse(C.begin(),C.end());
168     while(C.size() > 1 && C.back() == 0)C.pop_back();
169     return C;
170 }
171 //大数除以大数
172 vector<int> BigInteger::div(vector<int> A,vector<int>
173     B){
174     int la = A.size(),lb = B.size();

```

```

172     int dv = la - lb; // 相差位数
173     vector<int> C(dv+1,0);
174     //将除数扩大,使得除数和被除数位数相等
175     reverse(B.begin(),B.end());
176     for(int i=0;i<dv;i++)B.push_back(0);
177     reverse(B.begin(),B.end());
178     lb = la;
179     for(int j=0;j<=dv;j++){
180         while(!cmp(A,B)){
181             A = sub(A,B);
182             C[dv-j]++;
183         }
184         B.erase(B.begin());
185     }
186     while(C.size()>1 && C.back() == 0)C.pop_back();
187     return C;
188 }
189 BigInteger BigInteger::operator + ( BigInteger & b){
190     BigInteger c;
191     c.s.clear();
192     c.s = add(s,b.s);
193     return c;
194 }
195
196 BigInteger BigInteger::operator - ( BigInteger & b) {
197     BigInteger c;
198     c.s.clear();
199     if(*this < b){
200         c.flag = -1;
201         c.s = sub(b.s,s);
202     }
203     else{
204         c.flag = 1;
205         c.s = sub(s,b.s);
206     }
207     return c;
208 }
209 BigInteger BigInteger::operator *(BigInteger & b){
210     BigInteger c;
211     c.s = mul(s,b.s);
212     return c;
213 }
214 BigInteger BigInteger::operator *(int& b){
215     BigInteger c;
216     c.s = mul(s,b);
217     return c;
218 }
219 BigInteger BigInteger::operator /(BigInteger & b){
220     BigInteger c;
221     if(*this < b){
222         c.s.push_back(0);
223     }
224     else{
225         c.flag = 1;
226         c.s = div(s,b.s);
227     }
228     return c;
229 }
230 BigInteger BigInteger::operator /(int b){
231     BigInteger c;
232     BigInteger t = b;
233     if(*this < t){
234         c.s.push_back(0);
235     }
236     else{

```

```

237         c.flag = 1;
238         c.s = div(s,b);
239     }
240     return c;
241 }
242 int main(){
243     BigInteger A,B;
244     cin>>A>>B;
245     cout<<A+B<<endl;
246     cout<<A-B<<endl;
247     cout<<A*B<<endl;
248     cout<<A/B<<endl;
249     return 0;
250 }

```

### 1.3.2 加法

```

1  vector<int> add(vector<int> &A,vector<int> &B)
2  {
3      vector<int> C;
4      int t=0;
5      for(int i=0;i<A.size();i++)
6      {
7          t+=A[i];
8          if(i<B.size()) t+=B[i];
9          C.push_back(t%10);
10         t/=10;
11     }
12     if(t) C.push_back(1); //进的最后一位
13     return C;
14 }
15

```

### 1.3.3 减法

```

1  #include <iostream>
2  #include <bits/stdc++.h>
3  #include <vector>
4  using namespace std;
5  const int N=1e5+10;
6
7  bool cmp(vector<int> &A,vector<int> &B)
8  {
9      if(A.size()!=B.size())return A.size()>B.size();
10     for(int i=A.size();i>0;i--){
11         if(A[i]!=B[i]) return A[i]>B[i];
12     }
13     return true;
14 }
15 vector<int> sub(vector<int> A,vector<int> B)
16 {
17     vector<int> C;
18     int t=0; //进位
19     for(int i=0;i<A.size()||i<B.size();i++)
20     {
21         if(i<A.size()) t=A[i]-t; //借一位
22         if(i<B.size()) t-=B[i];
23         C.push_back((t+10)%10);
24         if(t<0) t=1;
25         else t=0;
26     }

```

```

27 while(C.size()>1&&C.back()==0) C.pop_back();//删除
    前导零 C.back()最后一位的值; pop_back()删除最后
    一位;
28 return C;
29 }
30
31 int main()
32 {
33     string a,b;
34     cin>>a>>b;
35     vector<int> A,B;
36     for(int i=a.size()-1;i>=0;i--) A.push_back(a[i]-'0');
37     for(int i=b.size()-1;i>=0;i--) B.push_back(b[i]-'0');
38     if(cmp(A,B))
39     {
40         vector<int> C=sub(A,B);
41         for(int i=C.size()-1;i>=0;i--) cout<<C[i];
42     }
43     else
44     {
45         vector<int> C=sub(B,A);
46         cout<<"-";
47         for(int i=C.size()-1;i>=0;i--) cout<<C[i];
48     }
49 }

```

### 1.3.4 乘法

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 vector<int> mul(vector<int> &A, int b)
7 {
8     vector<int> C;
9     int t = 0;
10    for (int i = 0; i < A.size() || t; i++)
11    {
12        if (i < A.size()) t += A[i] * b;
13        C.push_back(t % 10);
14        t /= 10;
15    }
16
17    return C;
18 }
19
20
21 int main()
22 {
23     string a;
24     int B;
25     vector<int> A;
26     cin >> a >> B;
27     for (int i = a.size() - 1; i >= 0; i--) A.
        push_back(a[i] - '0');
28
29     auto C = mul(A, B);
30
31     for (int i = C.size() - 1; i >= 0; i--) cout <<
        C[i];
32     cout << endl;

```

```

33
34     return 0;
35 }

```

### 1.3.5 除法

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 vector<int> div(vector<int> &A, int b, int &r)
8 {
9     vector<int> C;
10    r = 0;
11    for (int i = A.size() - 1; i >= 0; i--)
12    {
13        r = r * 10 + A[i];
14        C.push_back(r / b);
15        r %= b;
16    }
17    reverse(C.begin(), C.end());
18    while (C.size() > 1 && C.back() == 0) C.pop_back();
19    ;
20    return C;
21 }
22
23 int main()
24 {
25     string a;
26     vector<int> A;
27
28     int B;
29     cin >> a >> B;
30     for (int i = a.size() - 1; i >= 0; i--) A.
        push_back(a[i] - '0');
31
32     int r;
33     auto C = div(A, B, r);
34
35     for (int i = C.size() - 1; i >= 0; i--) cout <<
        C[i];
36
37     cout << endl << r << endl;
38
39     return 0;
40 }

```

## 1.4 区间和

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef pair<int, int> PII;
4 const int N = 3e6 + 10;
5 int a[N], s[N];
6 vector<int> alls;
7 vector<PII> add, query;
8
9 int find(int x){
10     int l = 0, r = alls.size() - 1;
11     while(l<r){

```

```

12         int mid = (l + r) >> 1;
13         if(alls[mid] >= x) r = mid;
14         else l = mid + 1;
15     }
16     return r + 1;
17 }
18 int main()
19 {
20     int n, m ;
21     while(~scanf("%d%d", &n,&m)){
22         for(int i = 0; i < n; i++){
23             int x, c;
24             scanf("%d%d", &x, &c);
25             add.push_back({x, c});
26             alls.push_back(x);
27         }
28         for(int i = 0; i < m; i++){
29             int l, r;
30             scanf("%d%d", &l, &r);
31             query.push_back({l, r});
32             alls.push_back(l);
33             alls.push_back(r);
34         }
35         //排序, 去重
36         sort(alls.begin(), alls.end());
37         alls.erase(unique(alls.begin(), alls.end()), alls.end());
38         //处理输入
39         for(auto item : add){
40             int x = find(item.first);
41             a[x] += item.second;
42         }
43         //前缀和
44         for(int i = 1; i <= alls.size(); i++) s[i] = s[i-1] + a[i];
45         //处理询问
46         for(auto item : query){
47             int l = find(item.first), r = find(item.second);
48             printf("%d\n", (s[r] - s[l-1]));
49         }
50     }
51 }
52 }

```

## 1.5 set 技巧

更改排序方式

由于 set 默认从小到大排序, 若需要递减集合, 更改 cmp 函数

```

1 //一元组集合
2 struct cmp {
3     bool operator() (const int &a, const int &b) const
4     {
5         return lena > lenb;
6     }
7 };
8 set<int> s;
9 //集合
10 struct cmp {
11     bool operator() (const pair<int, int> &a, const
12     pair<int, int> &b) const {
13         int lena = a.second - a.first + 1;
14         int lenb = b.second - b.first + 1;

```

```

14         if (lena == lenb) return a.first < b.first;
15         return lena > lenb;
16     }
17 };
18 set<pair<int,int>, cmp> s;

```

## 2 图论

### 2.1 链式前向星

```

1 #include <iostream>
2 #include <cstring>
3 #include <cmath>
4 #include <string>
5 #include <vector>
6 #include <algorithm>
7 #include <cstdio>
8 #include <list>
9 #include <set>
10 #include <map>
11 #include <queue>
12 #include <stack>
13 #include <vector>
14 #include <algorithm>
15 #include <unordered_map>
16 using namespace std;
17 const int maxn = 1e5 + 10;
18 const int INF = 2147483647;
19 int n, t, m, k, num, root, dcc, c[maxn], uu[maxn*2], vv[
20     maxn*2];
21 int ans, dfn[maxn], low[maxn], du[maxn];
22 bool bridge[maxn], cut[maxn];
23 int sum;
24 struct Edge
25 {
26     int to;
27     int next;
28     int w;
29 };
30 Edge edge[maxn<<2];
31 int cnt = 0;
32 int head[maxn];
33 void add(int u, int v)
34 {
35     edge[cnt].to = v;
36     edge[cnt].next = head[u];
37     head[u] = cnt++;

```

### 2.2 树的 dfs 序

进入递归后和即将回溯前记录一次编号, 产生长度为 2N 的 \*\*dfs 序\*\*

\*\*特点\*\* : 两次出现 \*\*x\*\* 的位置之间为以 \*\*x\*\* 为根的子树的 dfs 序

```

1 void dfs(int u){
2     a[++m] = u;
3     vis[u] = 1;
4     for(int i = head[u]; ~i; i = edge[i].next){
5         int v = edge[i].to;
6         if(vis[v]) continue;
7         dfs(v);

```

```

8     }
9     a[++m] = u;
10 }

```

## 2.3 树的深度

```

1 void dfs(int u){
2     v[u] = 1;
3     for(int i = head[u]; ~i; i = edge[i].next){
4         int v = edge[i].to;
5         if(vis[v]) continue;
6         d[v] = d[u] + 1;
7         dfs(v);
8     }
9     a[++m] = u;
10 }

```

## 2.4 图的连通块划分

```

1 void dfs(int u){
2     vis[u] = 1;
3     for(int i = head[u]; ~i; i = edge[i].next){
4         int v = edge[i].to;
5         if(vis[v]) continue;
6         dfs(v);
7     }
8 }
9
10 for(int i = 1; i <= n; i++){ //加在main()中
11     if(!vis[i]){
12         cnt++; //没有访问过, 连通块次数加一, 继续dfs
13         dfs(i);
14     }
15 }

```

## 2.5 欧拉路

```

1 void enlur(){
2     st[++top] = 1;
3     while(top > 0){
4         int x = st[top], i = head[x];
5         while(i && vis[i]) i = nxt[i];
6         if(i){
7             st[++top] = ver[i];
8             vis[i] = vis[i ^ 1] = 1;
9             head[x] = nxt[i]; // 会修改head[x] 的值
10        }else{
11            top--;
12            ans[++t] = x;
13        }
14    }
15 }

```

## 2.6 强连通分量

```

1 const int N = 100010, M = 1000010;
2 int head[N], ver[M], nxt[M], dfn[N], low[N];
3 int st[N], ins[N], c[N];
4 vector<int> scc[N];

```

```

5 int n, m, tot, num, top, cnt;
6 void add(int x, int y){
7     ver[++tot] = y, nxt[tot] = head[x], head[x] = tot;
8 }
9 void tarjan(int x){
10    dfn[x] = low[x] = ++num;
11    st[++top] = x, ins[x] = 1;
12    for(int i=head[x];i;i=nxt[i]){
13        if(!dfn[ver[i]]){
14            tarjan(ver[i]);
15            low[x] = min(low[x], low[ver[i]]);
16        }else if(ins[ver[i]]) //有作用的横叉边以及返祖边
17            low[x] = min(low[x], dfn[ver[i]]);
18    }
19    if(dfn[x] == low[x]){//x可以作为这个连通分量的入口
20        cnt++;int y;
21        do{
22            y = st[top--], ins[y] = 0;
23            c[y] = cnt, scc[cnt].push_back(y);
24        }while(x != y);
25    }
26 }

```

## 2.7 点分治

求是否存在路径权值和为 k 的路径

```

1 const int N = 10000 + 5;
2 const int M = 20010;
3 int head[N], ver[M], edge[M], nxt[M];
4 int sz[N], del[N], dis[N], st[N], q[N], cnt, top;
5 int query[101], res[N];
6 bool has[10000010];
7 int n, m, tot, all, core, min_size;
8 void add(int x, int y, int z){
9     ver[++tot] = y, nxt[tot] = head[x], edge[tot] = z,
10    head[x] = tot;
11 }
12 void getsize(int x, int fa){
13     sz[x] = 1;
14     int mx = 0;
15     for(int i=head[x];i;i=nxt[i]){
16         int y = ver[i];
17         if(y == fa || del[y]) continue;
18         getsize(y, x);
19         mx = max(mx, sz[y]);
20         sz[x] += sz[y];
21     }
22     mx = max(mx, all - sz[x]);
23     if(mx < min_size) {min_size = mx, core = x;}
24 }
25 void getdis(int x, int fa){
26     if(dis[x] <= 10000000)
27         st[++cnt] = x, q[++top] = x;
28     for(int i=head[x];i;i=nxt[i]) {
29         int y = ver[i];
30         if(del[y] || y == fa) continue;
31         dis[y] = dis[x] + edge[i];
32         getdis(y, x);
33     }
34 }
35 void get(int x){
36     cnt = top = 0;
37     del[x] = 1;
38     dis[x] = 0;

```



```

38 has[dis[x]] = true;
39 q[++top] = x;
40 for(int i=head[x];i;i=nxt[i]){
41     int y = ver[i];
42     if(del[y]) continue;
43     dis[y] = edge[i];
44     cnt = 0;
45     getdis(y, x);
46     for(int j=1;j<=cnt;j++){
47         for(int k=1;k<=m;k++){
48             if(query[k] >= dis[st[j]]){
49                 res[k] |= has[query[k] - dis[st[j]]];
50             }
51         }
52     }
53     for(int j=1;j<=cnt;j++) has[dis[st[j]]] = true;
54 }
55 for(int i=1;i<=top;i++) has[dis[q[i]]] = false;
56 for(int i=head[x];i;i=nxt[i]){
57     int y = ver[i];
58     if(del[y]) continue;
59     min_size = inf; all = sz[y];
60     getSize(y, 0);
61     getSize(core, 0);
62     get(core);
63 }
64 }
65 int main(){
66     scanf("%d%d", &n, &m);
67     for(int i=1;i<n;i++){
68         int x, y, z;scanf("%d%d%d", &x, &y, &z);
69         add(x, y, z);
70         add(y, x, z);
71     }
72     for(int i=1;i<=m;i++) scanf("%d", &query[i]);
73     min_size = inf;
74     all = n;
75     core = 0;
76     getSize(1, 0);
77     getSize(core, 0);
78     get(core);
79     for(int i=1;i<=m;i++) puts(res[i] ? "AYE" : "NAY");
80     return 0;
81 }

```

## 2.8 树的重心

//树的重心可以通过简单的两次搜索求出 //实际上这两步操作可以在一次遍历中解决。对节点  $u$  的每一个儿子  $v$  递归处理，然后以更新的子节点子树节点数最大值，处理完所有子结点后，判断  $u$  是否为重心。

```

1 void dfs(int u, int pa) { //dfs找重心
2     son[u] = 1;
3     int res = 0;
4     for (int i = head[u]; ~i; i = edge[i].next) {
5         int v = edge[i].to;
6         if (v == pa) continue;
7         dfs(v, u);
8         son[u] += son[v];
9         res = max(res, son[v]);
10    }

```

```

11    res = max(res, n - son[u]);
12    if (res < siz) {
13        ans = u;
14        siz = res;
15    }
16 }
17 int getCenter(int x) {
18     ans = 0;
19     siz = INF;
20     dfs(x, -1);
21     return ans;
22 }

```

## 2.9 树的直径

### 2.9.1 两次 dfs

复杂度  $O(2n)$ ，不能处理负权边，能找出两端位置，bfs 可以记录路径

```

1 //dfs
2 typedef ll long long;
3 const int maxn=1e5+5;
4 const int INF=0x3f3f3f3f;
5
6 struct Edge
7 {
8     int v,l;
9     int next;
10 } edge[maxn<<2];
11
12 int vit[maxn],d[maxn];
13 int head[maxn],k;
14 int node,ans;
15
16 void init()
17 {
18     k=0;
19     memset(head,-1,sizeof(head));
20 }
21
22 void addedge(int u,int v,int l)
23 {
24     edge[k].v=v;
25     edge[k].l=l;
26     edge[k].next=head[u];
27     head[u]=k++;
28
29     edge[k].v=u;
30     edge[k].l=l;
31     edge[k].next=head[v];
32     head[v]=k++;
33 }
34 void dfs(int u,int t)
35 {
36     for(int i=head[u]; i!=-1; i=edge[i].next)
37     {
38         int v=edge[i].v;
39         if(vit[v]==0)
40         {
41             vit[v]=1;
42             d[v]=t+edge[i].l;
43             if(d[v]>ans)
44             {
45                 ans=d[v];

```

```

46         node=v;
47     }
48     dfs(v,d[v]);
49 }
50 }
51 }
52
53 memset(vit,0,sizeof(vit)); //加在主函数里
54 vit[1]=1;
55 ans=0;
56 dfs(1,0);
57
58 memset(vit,0,sizeof(vit));
59 vit[node]=1;
60 ans=0;
61 dfs(node,0);
62
63 //bfs
64 const int N = 200010;
65 vector<pair<int,int> > v[N];
66 int n, pre[N], vis[N], mark[N];
67 ll d[N];
68 int bfs(int s){
69     queue<int> q;
70     q.push(s);
71     memset(vis, 0, sizeof vis);
72     memset(d, 0, sizeof d);
73     memset(pre, 0, sizeof pre);
74     vis[s] = d[s] = 0;
75     int t = s;
76     while(q.size()){
77         int x = q.front();
78         q.pop();
79         if(d[x] > d[t])***
80             t = x;
81         vis[x] = 1;
82         for (int i = 0; i < v[x].size();i++){
83             int y = v[x][i].first;
84             if(vis[y])
85                 continue;
86             d[y] = d[x] + v[x][i].second;
87             pre[y] = x;****
88             q.push(y);
89         }
90     }
91     return t;
92 }
93 int main(){
94     scanf("%d", &n);
95     for (int i = 1,x, y, z; i < n;i++){
96         scanf("%d%d%d", &x, &y, &z);
97         v[x].push_back({y, z});
98         v[y].push_back({x, z});
99     }
100     int s = bfs(1);
101     int t = bfs(s);
102     int p = t, np = pre[t];
103     ll len = d[t];
104     cout << len << endl;
105     while(p != s){
106         mark[p] = 1;
107         p = pre[p];
108         mark[p] = 1;
109     }
110     p = t;

```

```

111     return 0;
112 }

```

## 2.9.2 树形 dp

复杂度  $O(n)$ ，能处理负权边，不能找出两端位置

```

1 void dp(int u)
2 {
3     vis[u] = 1;
4     for (int i = head[u]; ~i; i = edge[i].next)
5     {
6         int v = edge[i].to;
7         if(vis[v]) continue;
8         dp(v);
9         ans = max(ans, d[u] + d[v] + edge[i].w);
10        d[u] = max(d[u], d[v] + edge[i].w);
11    }
12 }
13
14 dp(k); //主函数里
15 ans=2*sum-ans; //sum为所有边权的和

```

## 2.10 拓扑排序

```

1 // poj3687
2 #include <iostream>
3 #include <cstdio>
4 #include <cstdlib>
5 #include <cstring>
6 using namespace std;
7
8 const int maxn = 205;
9
10 bool g[maxn][maxn];
11 bool vis[maxn];
12 int in[maxn];
13 int ans[maxn];
14 int n, m;
15
16 void input()
17 {
18     scanf("%d%d", &n, &m);
19     memset(g, 0, sizeof(g));
20     memset(in, 0, sizeof(in));
21     for (int i = 0; i < m; i++)
22     {
23         int a, b;
24         scanf("%d%d", &a, &b);
25         a--;
26         b--;
27         if (!g[b][a])
28             in[a]++;
29         g[b][a] = true;
30     }
31 }
32
33 void work()
34 {
35     memset(vis, 0, sizeof(vis));
36     for (int i = n; i > 0; i--)
37     {
38         int q = -1;

```

```

39     for (int j = n - 1; j >= 0; j--)
40         if (!vis[j] && in[j] == 0)
41         {
42             q = j;
43             break;
44         }
45     if (q == -1)
46     {
47         printf("-1\n");
48         return;
49     }
50     vis[q] = true;
51     ans[q] = i;
52     for (int j = 0; j < n; j++)
53         if (g[q][j])
54             in[j]--;
55 }
56 printf("%d", ans[0]);
57 for (int i = 1; i < n; i++)
58     printf(" %d", ans[i]);
59 putchar('\n');
60 }
61
62 int main()
63 {
64     int t;
65     scanf("%d", &t);
66     while (t--)
67     {
68         input();
69         work();
70     }
71     return 0;
72     //system("pause");
73 }

```

```

1  #include <iostream>
2  #include <cstring>
3  #include <cmath>
4  #include <string>
5  #include <vector>
6  #include <algorithm>
7  #include <cstdio>
8  #include <queue>
9  using namespace std;
10 const int maxn = 1e5 + 10;
11 const int INF = 2147483647;
12 int n, m, k, num, root, dcc, c[maxn], uu[maxn*2], vv[
    maxn*2];
13 int ans, dfn[maxn], low[maxn], du[maxn];
14 bool bridge[maxn], cut[maxn];
15 struct Edge
16 {
17     int to;
18     int next;
19     int w;
20 };
21 Edge edge[maxn<<2];
22 int cnt = 0;
23 int tot;
24 int head[maxn], deg[maxn], a[maxn];
25 void add(int u, int v)
26 {
27     edge[cnt].to = v;
28     edge[cnt].next = head[u];

```

```

29     head[u] = cnt++;
30 }
31
32 void topsort() {
33     queue<int> q;
34     for (int i = 1; i <= n; i++)
35         if (deg[i] == 0) q.push(i);
36     while (q.size()) {
37         int x = q.front(); q.pop();
38         a[++tot] = x;
39         for (int i = head[x]; ~i; i = edge[i].next) {
40             int v = edge[i].to;
41             if (--deg[v] == 0) q.push(v);
42         }
43     }
44 }
45 int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
46 int main() {
47     int T, icase = 1;
48     scanf("%d", &T);
49     while (T--) {
50         cin >> n >> m;
51         cnt = 0;
52         tot = 0;
53         memset(head, -1, sizeof(head));
54         memset(deg, 0, sizeof(deg));
55         for (int i = 1; i <= m; i++) {
56             int x, y;
57             scanf("%d%d", &x, &y);
58             add(x, y);
59             deg[y]++;
60         }
61         topsort();
62         for (int i = 1; i <= tot; i++) {
63             printf("%d ", a[i]);
64         }
65         cout << endl;
66     }
67 }

```

## 2.11 dijkstra

1. 在稠密图: 朴素版 dijkstra  $O(n^2)$  2. 在稀疏图: 堆优化版 dijkstra  $O(m \log n)$

### 2.11.1 朴素算法

```

1 void Dijkstra(int src){
2     for (int i = 1; i <= n; ++i)
3         d[i] = INF;
4     d[src] = 0;
5     memset(vis, 0, sizeof(vis));
6     for (int i = 1; i <= n; ++i) {
7         int u = -1;
8         for (int j = 1; j <= n; ++j) if (!vis[j]) {
9             if (u == -1 || d[j] < d[u]) u = j;
10        }
11        vis[u] = 1;
12        for (int j = 1; j <= n; ++j) if (!vis[j]) {
13            int tmp = d[u] + w[u][j];
14            if (tmp < d[j]) d[j] = tmp;
15        }
16    }
17 }

```

### 2.11.2 堆优化

```

1 void Dijkstra(int src){
2     for(int i=1; i<=n; ++i)
3         d[i] = INF;
4     d[src] = 0;
5     memset(vis, 0, sizeof(vis));
6     for(int i=1; i<=n; ++i){
7         int u=-1;
8         for(int j=1; j<=n; ++j)if(!vis[j]){
9             if(u==-1 || d[j]<d[u]) u=j;
10        }
11        vis[u] = 1;
12        for(int j=1; j<=n; ++j)if(!vis[j]){
13            int tmp = d[u] + w[u][j];
14            if(tmp<d[j]) d[j] = tmp;
15        }
16    }
17 }

```

## 2.12 SPFA

### 2.12.1 求最短路

```

1 void spfa(int u)
2 {
3     memset(d, 0x3f, sizeof(d)), memset(vis, 0, sizeof(vis));
4     d[u] = 0, vis[u] = 1, q.push(u);
5     while(!q.empty()){
6         int x = q.front(); q.pop();
7         vis[x] = 0;
8         for(int i = head[u]; ~i; i = edge[i].next){
9             int y = edge[i].to, z = edge[i].w;
10            if(d[y] > d[x] + z){
11                d[y] = d[x] + z;
12                if(!vis[y]) q.push(y), vis[y] = 1;
13            }
14        }
15    }
16 }

```

### 2.12.2 判断负环

```

1 //设res(x)表示从1~x的最短路径包含的边数，若发现res(x)≥n
2 //，则图中有负环。
3 bool spfa()
4 {
5     memset(d, 0x3f, sizeof(d)), memset(vis, 0, sizeof(vis));
6     d[1] = 0, vis[1] = 1, res[1] = 0;
7     q.push(1);
8     while(!q.empty()){
9         int x = q.front(); q.pop();
10        vis[x] = 0;
11        for(int i = head[x]; ~i; i = edge[i].next){
12            int y = edge[i].to, z = edge[i].w;
13            if(d[y] > d[x] + z){
14                d[y] = d[x] + z;
15                res[y] = res[x] + 1;
16                if(res[y] >= n) return true;
17                if(!vis[y]){
18                    q.push(y), vis[y] = 1;
19                }
20            }
21        }
22    }
23 }

```

```

18     }
19 }
20 }
21 }
22 }
23 return false;
24 }

```

### 2.12.3 差分约束

```

1 bool spfa()
2 {
3     for(int i = 0; i <= n; i++){
4         d[i] = INF;
5         vis[i] = 0;
6     }
7     d[1] = 0, vis[1] = 1, res[1] = 0;
8     q.push(1);
9     while(!q.empty()){
10        int x = q.front(); q.pop();
11        vis[x] = 0;
12        for(int i = head[x]; ~i; i = edge[i].next){
13            int y = edge[i].to, z = edge[i].w;
14            if(d[y] > d[x] + z){ //要求最长的距离应求最短路
15                d[y] = d[x] + z;
16                res[y] = res[x] + 1;
17                if(!vis[y]){
18                    if(res[y] >= n) return true;
19                    q.push(y), vis[y] = 1;
20                }
21            }
22        }
23    }
24    return false;
25 }
26
27 for(int i = 1; i <= m; i++){ //距离不大于
28     scanf("%d%d%d", &u, &v, &w);
29     add(u, v, w);
30 }
31 for(int i = 1; i <= k; i++){ //距离不小于
32     scanf("%d%d%d", &u, &v, &w);
33     add(v, u, -w);
34 }
35 }

```

## 2.13 Floyd

### 2.13.1 求关系闭包 (递闭包)

```

1 #include <iostream>
2 #include <cstring>
3 #include <cmath>
4 #include <string>
5 #include <vector>
6 #include <algorithm>
7 #include <cstdio>
8 #include <queue>
9 #include <set>
10 using namespace std;
11 const int maxn = 1e4 + 10;
12 const int INF = 2147483647;
13 int n, t, m;

```

```

14 int ans, sum, d[maxn][maxn]; //Floyd一般用邻接矩阵存
   图
15 int main()
16 {
17     cin>>n>>m;
18     int u, v;
19     for(int i = 1; i <= n; i++)
20         for(int j=1; j<= n;j++)
21             d[i][j]=INF;
22     for(int i = 1; i <= m; i++){
23         scanf("%d%d", &u, &v);
24         d[u][v] = 1; //u胜v为1
25         d[v][u] = -1; //v负u为-1
26     }
27     for(int k = 1;k <= n; k++)
28         for(int i = 1; i <= n; i++){
29             for(int j=1; j <= n; j++){
30                 if(d[i][k] == d[k][j] && (d[i][k] == 1
31                     || d[i][k] == -1)) //求关系闭包, 若
32                     //若A→B, B→C, 则A→C
33                     d[i][j] = d[i][k];
34             }
35         }
36     for(int i = 1; i <= n; i++){
37         sum = 0;
38         for(int j = 1; j <= n; j++){
39             if(d[i][j]!=INF) sum++;
40         }
41         if(sum == n-1) ans++;
42     }
43     cout << ans << endl;
44     //system("pause");
45     return 0;
46 }

```

### 2.13.2 多源最短路

```

1 #include <iostream>
2 #include <cstring>
3 #include <cmath>
4 #include <string>
5 #include <vector>
6 #include <algorithm>
7 #include <cstdio>
8 #include <queue>
9 #include <set>
10 using namespace std;
11 const int maxn = 1e4 + 10;
12 const int INF = 2147483647;
13 int n, t, m;
14 int ans, sum, d[maxn][maxn]; //Floyd一般用邻接矩阵存
   图
15 int main()
16 {
17     cin>>n>>m;
18     int u, v;
19     for(int i = 1; i <= n; i++)
20         for(int j=1; j<= n;j++)
21             d[i][j]=INF;
22     for(int i = 1; i <= m; i++){
23         scanf("%d%d", &u, &v);
24         d[u][v] = 1; //u胜v为1
25         d[v][u] = -1; //v负u为-1
26     }
27     for(int k = 1;k <= n; k++)

```

```

28     for(int i = 1; i <= n; i++)
29         for(int j=1; j <= n; j++){
30             if(d[i][k] == d[k][j] && (d[i][k] == 1
31                 || d[i][k] == -1)) //求关系闭包, 若
32                 //若A→B, B→C, 则A→C
33                 d[i][j] = d[i][k];
34             }
35         }
36     for(int i = 1; i <= n; i++){
37         sum = 0;
38         for(int j = 1; j <= n; j++){
39             if(d[i][j]!=INF) sum++;
40         }
41         if(sum == n-1) ans++;
42     }
43     cout << ans << endl;
44     //system("pause");
45     return 0;
46 }

```

## 2.14 LCA(最近公共祖先)

```

1 /*- **定义**：两个节点的最近公共祖先，就是这两个点的公
   共祖先里面，离根最远的那个。
2
3 - **性质**：1.两点最近公共祖先必在树上两点间最短路上
4
5           2.d是树上两点间距离，h为点到树根的距离
6           $$
7           d(u,v) = h(u) + h(v) - 2h(LCA(u,v))
8           $$
9
10        **Tarjan算法**
11
12        - 使用并查集对“向上标记法”的优化
13        - 三类节点：1.vis[x] == 2 已访问并回溯
14                      2.vis[x] == 1 已开始递归但未回溯，正在访问的节点
15                      x以及x的祖先
16                      3.vis[x] == 0 尚未访问的节点*/
17 int getf(int x) //寻找父亲
18 {
19     return f[x] == x ? x : f[x] = getf(f[x]);
20 }
21 void Tarjan(int u)
22 {
23     vis[u] = 1;
24     int to;
25     for (int i = head[u]; ~i; i = edge[i].next)
26     {
27         to = edge[i].to;
28         if (vis[to]) continue;
29         d[to] = d[u] + edge[i].w;
30         Tarjan(to);
31         f[to] = u;
32     }
33     for (int i = 0; i < query[u].size(); i++)
34     {
35         int to = query[u][i], id = quert_id[u][i];
36         if (vis[to] == 2)
37         {
38             int lca = getf(to);
39             ans[id] = min(ans[id], d[u] + d[to] - 2 * d
40                 [lca]);
41         }
42     }
43 }

```

```

41     }
42     vis[u] = 2; //一回溯, 标记为2
43 }
44
45 /*小技巧: 存储离线一次性读入
46 $$
47 vector<int> query[N]
48 $$
49 */
50
51 vector<int> query[maxn], quert_id[maxn];
52 void add_query(int x, int y, int id) //只要有关, 就要
    加入询问
53 {
54     query[x].push_back(y), quert_id[x].push_back(id);
55     query[y].push_back(x), quert_id[y].push_back(id);
56 }
57 for (int i = 1; i <= m; ++i) //读入询问!!! 学会方法
58 {
59     int x, y;
60     scanf("%d%d", &x, &y);
61     if(x == y) ans[i] = 0;
62     else {
63         add_query(x,y,i);
64         ans[i] = 1 << 30;
65     }
66 }

```

## 2.15 二分图匹配 (染色法)

```

1  /*定理: 一张图是二分图, 当且仅当图中不存在奇环(长度为
    奇数的环)。
2  方法: 121212...染色, 没有矛盾就是二分图
3  */
4  void dfs(int u, int c)
5  {
6      color[u] = c;
7      for(int i = head[u]; ~i; i = edge[i].next){
8          int v = edge[i].to;
9          if(!color[v]){
10             dfs(v,3-c);
11         }
12     }
13 }
14 int main()
15 {
16     cin >> t;
17     while(t--){
18         scanf("%d%d", &n,&m);
19         for(int i = 1; i <= n; i++)
20             color[i]=0,head[i]=-1; //不要用memset, 会超
                时!!!
21         cnt = 0;
22         int u, v;
23         for(int i = 0; i < m; i++){
24             scanf("%d%d",&u,&v);
25             add(u,v);
26             add(v,u);
27         }
28         dfs(1,1);
29         k = 0, l = 0;
30         for(int i = 1; i <= n; i++){
31             if(color[i] == 1) a[++k] = i;
32             else b[++l] = i;

```

```

33     }
34     bool flag;
35     if(k <= n/2) flag = 1;
36     else flag = 0;
37     if(flag){
38         printf("%d\n",k);
39         for(int i = 1; i <=k; i++) printf("%d ",a[i]
40             ]);
41         cout<<endl;
42     }
43     else
44     {
45         printf("%d\n",l);
46         for(int i = 1; i <=l; i++) printf("%d ",b[i]
47             ]);
48         cout<<endl;
49     }
50     }
51     //system("pause");
52     return 0;
53 }
54 //也可用bfs
55 bool bfs1(int s)
56 {
57     queue<int> q;
58     color[s] = 1;
59     q.push(s);
60     while(q.size()){
61         int u = q.front();
62         q.pop();
63         for(int i = head[u]; ~i; i = edge[i].next){
64             int v = edge[i].to;
65             if(!color[v]){
66                 color[v] = 3-color[u];
67                 q.push(v);
68             }
69             else if(color[v] == color[u] ) return false
70                 ;
71         }
72     }
73     return true;
74 }

```

## 2.16 二分图最大匹配 (匈牙利算法)

```

1  /*最大匹配: "任意两条边都没有公共端点" 的边的集合被称为图
    的一组**匹配**, 在二分图中, 包含边最 多的一组匹配被称
    为二分图的最大匹配。5
2
3  寻找增广路: 1.y本身是非匹配点
4              2.y已与左部点x1匹配, 但从x1出发可找到另一个右
                部点y1匹配, 此时x~y~x1~y1是一条增广路
5  */
6  bool dfs2(int u)
7  {
8      for(int i = head[u]; ~i; i = edge[i].next){
9          int v = edge[i].to;
10         if(!vis[v]){
11             vis[v] = 1;
12             if(!match[v] || dfs2(match[v])){
13                 match[v] = u;
14                 return true;
15             }

```

```

16     }
17 }
18 return false;
19 }
20 for(int i = 1; i <= n; i++){
21     for(int j = 1; j <= n; j++) vis[j] = 0;
22     if(dfs2(i)) ans++;
23 }
    
```

```

20 if(u == v) continue;
21 //主函数中
22 for(int i = 1; i <= n; i++){
23     if(!dfn[i]) root = i, tarjan2(i);
24 }
25 for(int i = 1; i <= n; i++){
26     if(cut[i]) printf("%d", i);
27 }
    
```

## 2.17 Tarjan 求割边

```

1  /*时间戳：在图的深度优先遍历过程中，按照每个节点第一次被访
   问的时间顺序，依次给予1~n的数字标记，该标记被称为“时
   间戳”。***dfn[i]***表示
2
3  subtree(x)表示搜索树中以x为根的子树结点的集合。
4
5  low(x)（追溯值）**表示顶点x及其子树中的点，通过非搜索树上
   的边，能够回溯到的最早的点(dfn最小)的dfn值(但不能连接
   x与其父节点的边)。
6
7  割边判定法则：dfn(x)<low(y)
8  */
9  void tarjan1(int u, int in_edge)
10 {
11     dfn[u] = low[u] = ++num;
12     for(int i = head[u]; ~i; i = edge[i].next){
13         int v = edge[i].to;
14         if(!dfn[v]){
15             tarjan1(v, i);
16             low[u] = min(low[v], low[u]);
17             if(low[v] > dfn[u]){
18                 bridge[i] = bridge[i ^ 1] = true;
19             }
20         }
21         else if(i != (in_edge ^ 1))
22             low[u] = min(low[u], dfn[v]);
23     }
24 }
    
```

## 2.18 Tarjan 求割点

```

1  void tarjan2(int u)
2  {
3     dfn[u] = low[u] = ++num;
4     int flag = 0;
5     for(int i = head[u]; ~i; i = edge[i].next){
6         int v = edge[i].to;
7         if(!dfn[v]){
8             tarjan2(v);
9             low[u] = min(low[v], low[u]);
10            if(low[v] >= dfn[u]){
11                flag++;
12                if(flag > 1 || u != root)cut[u] =
                    true;
13            }
14        }
15        else low[u] = min(low[u], dfn[v]);
16    }
17 }
18 //加边中
    
```

## 2.19 网络流

### 2.19.1 最大流

整个网络流量最大，流函数  $f$  为最大流  
点上有权值，都要拆点

```

1  /**增广路算法**
2  /*O(nm2) 一般可以处理103~104 规模的网络
3  - 各条边剩余流量大于0，为一条增广路，不断用/bfs求增广路
4
5  - O(nm^2),1000~10000
6  - 为了存反向边，邻接表成对存储
7  - (x,y) 减小e, (y,x)增大e
8  */
9  #include <bits/stdc++.h>
10 using namespace std;
11 typedef long long ll;
12 const int N = 1010, M = 10010, inf = 1 << 29;
13 int head[N], next[N], ver[M], edge[M], v[N], incf[N],
    pre[N];
14 int n, m, s, t, tot, maxflow;
15 void add(int x, int y, int z){
16     ver[++tot] = y; edge[tot] = z; next[tot] = head[x];
17     head[x] = tot;
18     ver[++tot] = x; edge[tot] = 0; next[tot] = head[y];
19     head[y] = tot;
20 }
21 bool bfs(){
22     memset(v, 0, sizeof(v));
23     queue<int> q;
24     q.push(s); v[s] = 1;
25     incf[s] = inf; //增广路各边最小剩余流量
26     while(q.size()){
27         int x = q.front();
28         q.pop();
29         for(int i = head[x]; i; i = next[i]){
30             if(edge[i]){
31                 int y = ver[i];
32                 if(v[y]) continue;
33                 incf[y] = min(incf[x], edge[i]); //最小
                    剩余容量作为此条增广路流量
34                 pre[y] = i;
35                 q.push(y); v[y] = 1;
36                 if(y == t) return 1;
37             }
38         }
39     }
40     return 0;
41 }
42 void update(){//更新增广路和反向边剩余流量
43     int x = t;
44     while(x != s){
45         int i = pre[x];
    
```



```

46     edge[i] -= incf[t];
47     edge[i^1] += incf[t]; // “成对存储”xor 1 ,找到反
        向的容量
48     x = ver[i^1];
49 }
50     maxflow += incf[t];
51 }
52
53 int main(){
54     while(cin >> n >> m){
55         memset(head, 0, sizeof(head));
56         s = 1; t = n; tot = 1; maxflow = 0;
57         for(int i = 0; i < m; i++){
58             int x, y, z;
59             scanf("%d%d%d", &x, &y, &z);
60             add(x, y, z);
61         }
62         while(bfs()) update();
63         cout << maxflow << endl;
64     }
65 }
66 /*
67 7 10
68 1 2 4
69 1 3 5
70 2 3 1
71 2 4 5
72 3 4 2
73 3 5 5
74 4 7 5
75 5 4 6
76 5 6 3
77 6 7 2
78 */

```

Dinic 算法 \*\* (更快,  $1e4 \ 1e5$ ) 1. BFS 求出节点层次, 构造分层图 2. 分层图 DFS 找增广路, 回溯时更新剩余容量, 每个点可以流向多条边, 还要剪枝

```

1 //O(n2m) 该算法求解二分图最大匹配的时间复杂度为O(mn--v)
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const int N = 5010, M = 30010, inf = 1 << 29;
6 int n,m,s,t,tot,maxflow;
7 int ver[M], next[M], head[N], edge[M], d[N];
8 queue<int> q;
9 void add(int x, int y, int z){
10     ver[++tot] = y; edge[tot] = z; next[tot] =
        head[x]; head[x] = tot;
11     ver[++tot] = x; edge[tot] = 0; next[tot] =
        head[y]; head[y] = tot;
12 }
13
14 bool bfs(){
15     memset(d,0,sizeof(d));
16     while(q.size()) q.pop(); //每一次生成分层图之前
        先清空栈
17     q.push(s), d[s] = 1;
18     while(q.size()){
19         int x = q.front(); q.pop();
20         for(int i = head[x]; i; i = next[i]){
21             if(edge[i] && !d[ver[i]]){d[ver[i]](与
                x相连的下一个点) 不为零, 说明已构成了分
                层图中的一层, 不满足条件
22             d[ver[i]] = d[x] + 1;

```

```

23         q.push(ver[i]);
24         //cout << "1" <<endl;
25         if(ver[i] == t) return 1; //可以到达汇
            点, 说明可以构成一个分层图
26     }
27 }
28 }
29 return 0;
30 }
31
32 int dinic(int x, int flow){
33     if(x == t) return flow;
34     int rest = flow, k;
35     for(int i = head[x]; i && rest; i = next[i])
36         //rest为什么不能等于0
37         if(edge[i] && d[ver[i]] == d[x] + 1){
38             k = dinic(ver[i], min(edge[i], rest));
39             if(!k) d[ver[i]] = 0; //剪枝, 去掉增广完
                毕的点(什么意思?)
40             edge[i] -= k;
41             edge[i^1] += k;
42             rest -= k;
43         }
44     return flow - rest; //相当于返回 k (已经找到的流
        流量)
45 }
46
47 int main(){
48     while(cin >> n >> m){
49         s = 1; t = n; tot = 1; maxflow = 0;
50         for(int i = 0; i < m; i++){
51             int x, y, z;
52             scanf("%d%d%d", &x, &y, &z);
53             add(x, y, z);
54         }
55         int flow = 0;
56         while(bfs())
57             while(flow = dinic(s, inf)) maxflow +=
                flow;
58         cout << maxflow << endl;
59     }
60     system("pause");
61 }

```

## 2.19.2 最小割

```

1 // poj 1966
2 // 题意: 无向图, 去掉点, 使不连通
3 //技巧: 点边转化
4 #include <bits/stdc++.h>
5 using namespace std;
6 typedef long long ll;
7 const int N = 5010, M = 30010, inf = 1e9;
8 int n,m,s,t,tot,maxflow,ans;
9 int ver[M], Next[M], head[N], edge[M], d[N];
10 queue<int> q;
11 int x[M], y[M];
12
13 void add(int x, int y, int z){
14     ver[++tot] = y; edge[tot] = z; Next[tot] = head[x]
        ]; head[x] = tot;
15     ver[++tot] = x; edge[tot] = 0; Next[tot] = head[y]
        ]; head[y] = tot;
16 }

```



```

17
18 bool bfs(){
19     memset(d,0,sizeof(d));
20     while(q.size()) q.pop(); //每一次生成分层图之前先清
        空栈
21     q.push(s), d[s] = 1;
22     while(q.size()){
23         int x = q.front(); q.pop();
24         for(int i = head[x]; i; i = Next[i]){
25             if(edge[i] && !d[ver[i]]){ //d[ver[i]] (与x
                相连的下一个点) 不为零, 说明已构成了分层图
                中的一层, 不满足条件
26                 d[ver[i]] = d[x] + 1;
27                 q.push(ver[i]);
28                 //cout << "1" << endl;
29                 if(ver[i] == t) return 1; //可以到达汇点,
                    说明可以构成一个分层图
30             }
31         }
32     }
33     return 0;
34 }
35
36 int dinic(int x, int flow){
37     if(x == t) return flow;
38     int rest = flow, k;
39     for(int i = head[x]; i && rest; i = Next[i]) //
        rest为什么不能等于0
40         if(edge[i] && d[ver[i]] == d[x] + 1){
41             k = dinic(ver[i], min(edge[i], rest));
42             if(!k) d[ver[i]] = 0; //剪枝, 去掉增广完毕的
                点(什么意思?)
43             edge[i] -= k;
44             edge[i^1] += k;
45             rest -= k;
46         }
47     return flow - rest; //相当于返回 k (已经找到的流流
        量)
48 }
49
50 int main(){
51     while(cin >> n >> m){
52         char str[100];
53         for(int i=1;i<=m;++i)
54             {
55                 scanf("%s",&str); int j;
56                 for(x[i]=0,j=1;str[j]!='\0';j++) x[i]=x[i]
                    *10+str[j]-'0';
57                 for(y[i]=0,j=1;str[j]!='\0';j++) y[i]=y[i]
                    *10+str[j]-'0';
58                 x[i]++; y[i]++;
59             }
60         ans = inf;
61         for(s = 1; s <= n; s++){
62             for(t = 1; t <= n; t++){
63                 if(s!=t){
64                     memset(head, 0, sizeof(head));
65                     tot = 1; maxflow = 0;
66                     int flow;
67                     for(int i = 1; i <= n; i++) {
68                         if(s == i || t == i) add(i, i+n,
                            inf);
69                         else add(i, i+n, 1);
70                     }
71                     for(int i = 1; i <= m; i++){

```

```

72         add(x[i]+n, y[i], inf);
73         add(y[i]+n, x[i], inf);
74     }
75     while(bfs())
76         while(flow = dinic(s, inf))
77             maxflow += flow;
78     ans = min(ans, maxflow);
79 }
80 if(ans == inf || n<=1) ans = n;
81 cout << ans << endl;
82 }
83 }

```

### 2.19.3 有向无环图最小路径覆盖

尽量少的不相交的简单路径, 每个顶点恰好覆盖一次

拆点,  $x, x+n, 1 \sim n$  为左部点,  $n+1 \sim 2n$  为右部点, 得到拆点二分图

定理: 最小路径点覆盖包含路径点条数 =  $n$  - 拆点二分图最大匹配数

```

1 /*
2 libre oj 6002 最小路径覆盖
3
4 **记录路径方法**: 用一个nex[]数组, 在增广成功时, nex[x]
    = to - n(下一个点); 然后遍历起点(没有进入的点,)
5 */
6 #include <bits/stdc++.h>
7 using namespace std;
8 typedef long long ll;
9 const int N = 5010, M = 30010, inf = 1e9;
10 int n,m,s,t,tot,maxflow,ans;
11 int ver[M], Next[M], head[N], edge[M], d[N];
12 queue<int> q;
13 int nex[N];
14
15 void add(int x, int y, int z){
16     ver[++tot] = y; edge[tot] = z; Next[tot] = head[x]
    ]; head[x] = tot;
17     ver[++tot] = x; edge[tot] = 0; Next[tot] = head[y]
    ]; head[y] = tot;
18 }
19
20 bool bfs(){
21     memset(d,0,sizeof(d));
22     while(q.size()) q.pop(); //每一次生成分层图之前先清
        空栈
23     q.push(s), d[s] = 1;
24     while(q.size()){
25         int x = q.front(); q.pop();
26         for(int i = head[x]; i; i = Next[i]){
27             if(edge[i] && !d[ver[i]]){ //d[ver[i]] (与x
                相连的下一个点) 不为零, 说明已构成了分层图
                中的一层, 不满足条件
28                 d[ver[i]] = d[x] + 1;
29                 q.push(ver[i]);
30                 if(ver[i] == t) return 1; //可以到达汇点,
                    说明可以构成一个分层图
31             }
32         }
33     }
34     return 0;
35 }
36

```

```

37 int dinic(int x, int flow){
38     if(x == t) return flow;
39     int rest = flow, k;
40     for(int i = head[x]; i && rest; i = Next[i]) //
        rest为什么不能等于0
41         if(edge[i] && d[ver[i]] == d[x] + 1){
42             k = dinic(ver[i], min(edge[i], rest));
43             if(!k) d[ver[i]] = 0; //剪枝, 去掉增广完毕的
                点(什么意思?)
44             edge[i] -= k;
45             edge[i^1] += k;
46             rest -= k;
47             if(k) nex[x] = ver[i] - n;
48         }
49     return flow - rest; //相当于返回 k (已经找到的流流
        量)
50 }
51
52 int main(){
53     while(cin >> n >> m){
54         s = 0, t = n + 1, ans = 0, tot = 1, maxflow
            = 0;
55         for(int i=1;i<=n;++i) {
56             add(s, i, 1); add(i+n, t, 1);
57         }
58         for(int i = 1; i <= m; i++) {
59             int x, y;
60             scanf("%d%d", &x,&y);
61             add(x, y+n, 1);
62         }
63         int flow = 0;
64         while(bfs())
65             while(flow = dinic(s, inf)) maxflow += flow
                ;
66         for(int i = head[t]; i; i = Next[i]) {
67             if(edge[i^1]){
68                 int p = ver[i] - n;
69                 while(p) printf("%d ", p), p = nex[p];
70                 puts("");
71             }
72         }
73         ans = n - maxflow;
74         cout << ans << endl;
75     }
76 }

```

### 3 数据结构

#### 3.1 并查集

```

1 int n,m,x,y;
2 int f[N],ronk[N];
3
4 void init()//建立森林
5 {
6     for(int i=0;i<n;i++)
7     {
8         f[i]=i;ronk[i]=0;
9     }
10 }
11
12
13 int getf(int x){
14     return f[x]==x? x:f[x]=getf(f[x]);

```

```

15 }
16
17 void unionset(int i,int j){
18     x=getf(i);
19     y=getf(j);
20     if(x==y) return;
21     else{
22         f[x]=y;
23     }
24 }
25
26 void unionset(int i,int j)//合并并查集, 根据秩的大小,
    小的连在大的上面
27 {
28     x=getf(i);
29     y=getf(j);
30     if(x==y) return;
31     if(ronk[x]>ronk[y])
32         f[y]=x;
33     else
34     {
35         f[x]=y;
36         if(ronk[x]==ronk[y])
37             ronk[x]++;
38     }
39 }
40
41 for (int i=1;i<=n;i++) num[getf(i)]++;//求元素所在集合
    元素个数
42 for (int i=1;i<=n;i++) cout<<num[getf(i)]<<' ';
43
44 void unionset(int i,int j)//求个数时误用, 会超时!!!
45 {
46     int x=getf(i);
47     int y=getf(j);
48     if(x==y) return;
49     if(ronk[x]>ronk[y]){
50         f[y]=x;
51         num[x]+=num[y];
52     }
53     else
54     {
55         f[x]=y;
56         if(ronk[x]==ronk[y])
57             ronk[x]++;
58         num[y]+=num[x];
59     }
60 }

```

#### 带权并查集

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 4000010;
4 int fa[N],to[N],sz[N],cnt,a[N],b[N],tot;
5 int n,m,k,x,y;
6 int ok[N];
7 int find(int x){
8     return x == fa[x]?x:fa[x] = find(fa[x]);
9 }
10 void merge(int x,int y){
11     int a = find(to[x]);
12     int b = find(to[y]);
13     if(a==b) return;
14     fa[b] = a;
15     sz[a] += sz[b]; sz[b] = 0;

```

```

16 }
17 void update(int x,int y){
18     sz[find(to[x])]--;
19     to[x] = ++cnt;
20     sz[cnt] = 1;
21     fa[cnt] = cnt;
22     merge(x,y);
23 }
24 int main(){
25     scanf("%d%d",&n,&m);cnt = n;
26     for(int i=1;i<=n;i++)fa[i] = i,to[i] = i,sz[i] = 1;
27     for(int i=1;i<=m;i++){
28         scanf("%d%d",&k,&x);if(k!=3)scanf("%d",&y);
29         if(k == 5){
30             a[++tot] = x;
31             b[tot] = y;continue;
32         }
33         if(k == 1)merge(x,y);
34         if(k == 2)update(x,y);
35         if(k == 4){
36             if(find(to[x]) == find(to[y]))printf("Yes\n");
37             else printf("No\n");
38         }
39         if(k == 3)printf("%d\n",sz[find(to[x])]-1);
40     }
41 }
42 for(int i=1;i<=tot;i++){
43     if(find(to[a[i]]) == find(to[b[i]])) ok[find(to[a[i]])] = 1;
44 }
45 int res = -1;
46 for(int i=1;i<=n;i++)
47     if(!ok[find(to[i])])res = max(res,sz[find(to[i])]);
48 cout<<res<<endl;
49 return 0;
50 }
51

```

可持久化并查集

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int inf = 0x3f3f3f3f;
5 #define dbg(x...) do { cout << "\033[32;1m" << #x << "
   -> "; err(x); } while (0)
6 void err() { cout << "\033[39;0m" << endl; }
7 template<class T, class... Ts> void err(const T& arg,
   const Ts&... args) { cout << arg << " "; err(args
   ...); }
8 const int N = 200000 + 5;
9
10 int n, m;
11 struct TreeNode{
12     int l, r;
13 };
14 int root[N];
15 class Union_Find{
16 public:
17     int fa[N*30], dep[N*30], tot;
18     TreeNode t[N*30];
19     void build(int &p, int l, int r){
20         p = ++tot;

```

```

21         if(l == r){ fa[p] = l; return; }
22         int mid = l + r >> 1;
23         build(t[p].l, l, mid);
24         build(t[p].r, mid+1, r);
25     }
26     void change(int last, int &p, int l, int r, int
   pos, int val){
27         p = ++tot;
28         t[p].l = t[last].l, t[p].r = t[last].r;
29         if(l == r){
30             fa[p] = val;
31             dep[p] = dep[last];
32             return;
33         }
34         int mid = (l + r) >> 1;
35         if(pos <= mid) change(t[last].l, t[p].l, l,
   mid, pos, val);
36         else change(t[last].r, t[p].r, mid + 1, r, pos
   , val);
37     }
38     int getIndex(int p, int l, int r, int pos){
39         if(l == r) return p;
40         int mid = l + r >> 1;
41         if(pos <= mid) return getIndex(t[p].l, l, mid,
   pos);
42         else return getIndex(t[p].r, mid+1, r, pos);
43     }
44     void add(int p, int l, int r, int pos){
45         if(l == r){
46             dep[p] ++;
47             return;
48         }
49         int mid = l + r >> 1;
50         if(pos <= mid) add(t[p].l, l, mid, pos);
51         else add(t[p].r, mid+1, r, pos);
52     }
53     // find 返回的是祖先节点的下标
54     int find(int p, int pos){
55         int index = getIndex(p, 1, n, pos);
56         if(fa[index] == pos)
57             return index;
58         else
59             return find(p, fa[index]);
60     }
61     void merge(int last, int &p, int x, int y){
62         int fax = find(p, x), fay = find(p, y);
63         if(fa[fax] == fa[fay]) return;
64         if(dep[fax] > dep[fay]) swap(fax, fay);
65         change(last, p, 1, n, fa[fax], fa[fay]);
66         if(dep[fax] == dep[fay]) add(p, 1, n, fa[fay])
   ;
67     }
68 }uf;
69
70 int main(){
71     scanf("%d%d", &n, &m);
72     uf.build(root[0], 1, n);
73     for(int i=1;i<=m;i++){
74         int op, x, y;
75         scanf("%d%d", &op, &x);
76         if(op == 1){
77             scanf("%d", &y);
78             root[i] = root[i-1];
79             uf.merge(root[i-1], root[i], x, y);
80         } else if(op == 2) root[i] = root[x];

```

```

81     else{
82         scanf("%d", &y);
83         root[i] = root[i-1];
84         int fax = uf.find(root[i], x);
85         int fay = uf.find(root[i], y);
86         if(uf.fa[fax] == uf.fa[fay]) puts("1");
87         else puts("0");
88     }
89 }
90
91 return 0;
92 }

```

### 3.2 分块

```

1  int n,m,x,y;
2  int a[N], sum[N], add[N], pos[N], L[N], R[N]; //sum是
    块内所有元素的和 add是每一块一致变化的更改值 pos是当
    前元素属于那一块
3
4  void create(ll n) { //分块
5      ll t = sqrt(n);
6      for(ll i = 1; i <= t; i++) {
7          L[i] = (i-1) * sqrt(n) + 1;
8          R[i] = i*sqrt(n);
9      }
10     if(R[t] < n) t++, L[t] = R[t-1] + 1, R[t] = n;
11     for(ll i = 1; i <= t; i++)
12         for(ll j = L[i]; j <= R[i]; j++) {
13             pos[j] = i;
14             sum[i] += a[j];
15         }
16 }
17
18 void update(ll l, ll r, ll d) { //更新
19     ll p = pos[l], q = pos[r];
20     if(p == q) {
21         for(ll i = l; i <= r; i++) a[i] += d;
22         sum[p] += d*(r-l+1);
23     }
24     else {
25         for(ll i = p + 1; i < q; i++) {
26             add[i] += d;
27         }
28         for(ll i = l; i <= R[p]; i++) {
29             a[i] += d;
30             sum[p] += d;
31         }
32         for(ll i = L[q]; i <= r; i++) {
33             a[i] += d;
34             sum[q] += d;
35         }
36     }
37 }
38
39 ll ask(ll l, ll r) { //询问区间和
40     ll ans = 0;
41     ll p = pos[l], q = pos[r];
42     if(p == q) {
43         for(ll i = l; i <= r; i++) ans += a[i] + add[p];
44     }
45     else {
46         for(ll i = p + 1; i < q; i++)

```

```

47         ans += sum[i] + add[i] * (R[i] - L[i] + 1);
48         for(ll i = l; i <= R[p]; i++)
49             ans += a[i] + add[p] * (R[i] - l + 1);
50         for(ll i = L[q]; i <= r; i++)
51             ans += a[i] + add[q] * (r - L[i] + 1);
52     }
53     return ans;
54 }

```

### 3.3 树状数组

```

1  int lowbit(int x) {
2      return x & -x;
3  }
4  void init(){//线性构造
5      for (int i = 1; i <= n; i++){
6          pre[i] = pre[i - 1] + a[i];
7          c[i] = pre[i] - pre[i - lowbit(i)];
8      }
9  }
10
11 int ask(int x) { //查询前缀和 区间和[l,r]:ask(r) - ask
    (l-1);
12     int ans = 0;
13     for(; x; x -= x & -x) ans += c[x];
14     return res;
15 }
16
17 void update(int x, int t) { //单点增加
18     while(x <= n) {
19         c[x] += t;
20         x += lowbit(x);
21     }
22 }

```

### 3.4 线段树

```

1  #include <iostream>
2  #include <iomanip>
3  #include <string>
4  #include <cstdio>
5  #include <cstdlib>
6  #include <algorithm>
7  #include <cstring>
8  #include <vector>
9  using namespace std;
10 const int size = 1e5+10;
11 int a[size];
12 struct SetmentTree
13 {
14     int l, r, dat;
15 }t[size * 4];
16
17 void BuildTree(int p, int l, int r) //线段树建树
18 {
19     t[p].l = l, t[p].r = r;
20     if(l == r){
21         t[p].dat = a[l];
22         return;
23     }
24     int mid = (l + r) >> 1;
25     BuildTree(p * 2, l, mid), BuildTree(p * 2 + 1, mid
    + 1, r);

```

```

26     t[p].dat = max(t[p * 2].dat, t[p * 2 + 1].dat);
27 }
28
29 void change(int p, int x, int v){ //单点修改
30     if(t[p].l == t[p].r){
31         t[p].dat = v;
32         return;
33     }
34     int mid = (t[p].l + t[p].r) >> 1;
35     if(x <= mid) change(p * 2, x, v);
36     else change(p * 2 + 1, x, v);
37     t[p].dat = max(t[p*2].dat, t[p*2+1].dat);
38 }
39
40 int ask(int p, int l, int r){ //区间查询
41     if(l <= t[p].l && r >= t[p].r) return t[p].dat;
42     int mid = (t[p].l + t[p].r) >> 1;
43     int val = -(1 >> 30);
44     if(l <= mid) val = max(val, ask(p*2, l, r));
45     if(r > mid) val = max(val, ask(p*2+1, l, r));
46     return val;
47 }
48 int main()
49 {
50     int n;
51     scanf("%d", &n);
52     for(int i = 1; i <= n; i++){
53         scanf("%d",&a[i]);
54     }
55     cout<<"111"<<endl;
56     BuildTree(1, 1, n);
57     int T;
58     scanf("%d",&T);
59     while(T--){
60         int x, v;
61         scanf("%d%d",&x,&v);
62         change(1, x, v);
63         cout << t[1].dat <<endl;
64     }
65     cout << ask(1, 1 ,3);
66     system("pause");
67 }
68
69 //建树
70
71 void build(int s, int t, int p) {
72     // 对 [s,t] 区间建立线段树,当前根的编号为 p
73     if (s == t) {
74         d[p] = a[s];
75         return;
76     }
77     int m = (s + t) / 2;
78     build(s, m, p * 2), build(m + 1, t, p * 2 + 1);
79     // 递归对左右区间建树
80     d[p] = d[p * 2] + d[(p * 2) + 1];
81 }
82
83 // 区间查询
84
85 //区间查询,比如求区间的总和(即)、求区间最大值/最小值
86 //等操作。
87
88 int getsum(int l, int r, int s, int t, int p) {
89     // [l,r] 为查询区间,[s,t] 为当前节点包含的区间,p 为当

```

```

前节点的编号
90 if (l <= s && t <= r)
91     return d[p]; // 当前区间为询问区间的子集时直接返回当前
前区间的和
92 int m = (s + t) / 2, sum = 0;
93 if (l <= m) sum += getsum(l, r, s, m, p * 2);
94 // 如果左儿子代表的区间 [l,m] 与询问区间有交集,则递归查
询左儿子
95 if (r > m) sum += getsum(l, r, m + 1, t, p * 2 + 1)
;
96 // 如果右儿子代表的区间 [m+1,r] 与询问区间有交集,则递归
查询右儿子
97 return sum;
98 }
99
100 //区间修改与懒惰标记
101
102 //区间修改(区间加上某个值):
103
104 void update(int l, int r, int c, int s, int t, int p)
{
105     // [l,r] 为修改区间,c 为被修改的元素的变化量,[s,t] 为
当前节点包含的区间,p
106     // 为当前节点的编号
107     if (l <= s && t <= r) {
108         d[p] += (t - s + 1) * c, b[p] += c;
109         return;
110     } // 当前区间为修改区间的子集时直接修改当前节点的值,然后
打标,结束修改
111     int m = (s + t) / 2;
112     if (b[p] && s != t) {
113         // 如果当前节点的懒标记非空,则更新当前节点两个子节点的
值和懒标记值
114         d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[
p] * (t - m);
115         b[p * 2] += b[p], b[p * 2 + 1] += b[p]; // 将标记
下传给子节点
116         b[p] = 0; // 清空当前节点的标记
117     }
118     if (l <= m) update(l, r, c, s, m, p * 2);
119     if (r > m) update(l, r, c, m + 1, t, p * 2 + 1);
120     d[p] = d[p * 2] + d[p * 2 + 1];
121 }
122
123 //区间查询(区间求和):
124
125 int getsum(int l, int r, int s, int t, int p) {
126     // [l,r] 为修改区间,c 为被修改的元素的变化量,[s,t] 为
当前节点包含的区间,p
127     // 为当前节点的编号
128     if (l <= s && t <= r) return d[p];
129     // 当前区间为询问区间的子集时直接返回当前区间的和
130     int m = (s + t) / 2;
131     if (b[p]) {
132         // 如果当前节点的懒标记非空,则更新当前节点两个子节点的
值和懒标记值
133         d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[
p] * (t - m),
134         b[p * 2] += b[p], b[p * 2 + 1] += b[p]; // 将标
记下传给子节点
135         b[p] = 0; // 清空当前节点的标记
136     }
137     int sum = 0;
138     if (l <= m) sum = getsum(l, r, s, m, p * 2);
139     if (r > m) sum += getsum(l, r, m + 1, t, p * 2 + 1)

```

```

140     ;
141     return sum;
142 }

```

### 3.5 bfs

#### 3.5.1 小猫爬山

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 1e2;
4 int c[maxn], cab[maxn], cnt, ans, n, w;
5
6 bool cmp(int a, int b){
7     return a > b;
8 }
9
10 void dfs(int now, int cnt){
11     if(cnt >= ans) return;
12     if(now == n + 1){
13         ans = min(cnt, ans);
14         return;
15     }
16     for(int i = 1; i <= cnt; i++){
17         if(cab[i] + c[now] <= w){
18             cab[i] += c[now];
19             dfs(now + 1, cnt);
20             cab[i] -= c[now]; //还原现场 (为了尝试将小猫
21                             //放在下一个缆车里)
22         }
23     }
24     cab[cnt + 1] = c[now];
25     dfs(now + 1, cnt + 1);
26     cab[cnt + 1] = 0; //还原现场 (不将这个猫放在这个地
27     //方)
28 }
29
30 int main()
31 {
32     while(cin >> n >> w){
33         for(int i = 1; i <= n; i++) cin >> c[i];
34         sort(c + 1, c + n + 1, cmp);
35         ans = n;
36         dfs(1, 0);
37         cout << ans << endl;
38     }
39     //system("pause");
40     return 0;
41 }

```

#### 3.5.2 八皇后

```

1 int n,x;
2 char g[N][N];
3 int col[N], dg[N], udg[N];
4 void dfs(int p) {
5     if(p == n) {
6         for(int i = 0; i < n; i++) {
7             puts(g[i]);
8         }
9         puts("");
10        return;
11    }

```

```

12    for(int i = 0; i < n; i++){
13        if(!col[i] && !dg[i+p] && !udg[n+i-p]) {
14            g[p][i] = 'Q';
15            col[i] = dg[i+p] = udg[n+i-p] = 1;
16            dfs(p+1);
17            dg[i+p] = udg[n+i-p] = col[i] = 0; //还原现场
18            g[p][i] = '.';
19        }
20    }
21 }
22
23 int main(){
24     scanf("%d", &n);
25     for(int i = 0; i < n; i++)
26         for(int j = 0; j < n; j++)
27             g[i][j] = '.';
28     dfs(0);
29 }

```

## 4 数论

### 4.1 筛质数

#### 4.1.1 线性筛

```

1 int prime[MAXN], v[MAXN], son[MAXN]; int m=0; //m表示现在
2 //筛出m个质数 son是最小质因数
3 void primes()
4 {
5     for(int i=2; i<N; i++){
6         if(v[i]==0) //如果v[i]为0, 说明 i 之前没有被筛到
7             //过, i 为质数
8         {
9             prime[++m] = i;
10            son[i] = i; //son[i] = j: i的最小质因数为
11            //j
12        }
13        for(int j = 1; j<=m; j++) //遍历小于 i 的所有质数
14        {
15            int t = prime[j]; //乘起来大于MAXN就跳出循环
16            if(t*i > N) break;
17            v[i*t] = 1; //标记 i*prime[j] 的最小质因数是
18            //prime[j]
19            son[i*t] = t;
20            if(i%t==0) break;
21        }
22    }
23 }

```

### 4.2 质因数分解

```

1 //p[]存所有的质因数, c[]存个数
2 void divide(int n){
3     m = 0;
4     for(int i=2; i<=sqrt(n); i++){
5         if(n % i == 0){
6             p[++m] = i, c[m] = 0;
7             while(n % i == 0) n /= i, c[m]++;
8         }
9     }
10    if(n > 1)
11        p[++m] = n, c[m] = 1;

```

12 }

### 4.3 最大质因数

```

1 void init()
2 {
3     for(int i = 2; i < N; i++)
4     {
5         if(prime[i] == 0)
6         {
7             for(int j = i; j < N; j += i)
8             {
9                 prime[j] = i;
10            }
11        }
12    }
13 }
```

### 4.4 正因数集合

```

1 vector<int> fac[500010];
2 for(int i=1;i<=n;i++)
3     for(int j=1;j<=n/i;j++)
4         fac[i*j].push_back(i);
```

## 4.5 逆元

### 4.5.1 费马小定理

```

1 //用快速幂，费马小定理
2 ll qpow(ll a,ll b) {
3     ll ans=1; a=a%mod;
4     while(b>0){
5         if(b%2==1) ans=(ans*a)%mod;
6         b=b/2; a=a*a%mod;
7     }
8     return ans;
9 }
10 ll inv(ll a) {
11     return qpow(a, mod-2);
12 }
```

### 4.5.2 线性逆推

```

1 typedef long long ll;
2 const int N = 1e5 + 5;
3 ll fac[N], inv[N], fac_inv[N];
4 void getFac(int n){
5     fac[0] = fac_inv[0] = 1;
6     for(int i=1;i<=n;i++) fac[i] = fac[i-1] * i % mod;
7     inv[1] = 1;
8     for(int i=2;i<=n;i++) inv[i] = 1ll * (mod-mod/i)*
9         inv[mod%i] % mod;
10    for(int i=1;i<=n;i++) fac_inv[i] = fac_inv[i-1] *
11        inv[i] % mod;
12 }
```

### 4.5.3 扩展欧几里得

```

1 typedef long long ll;
2 void extgcd(ll a,ll b,ll& d,ll& x,ll& y){
3     if(!b){ d=a; x=1; y=0;}
4     else{ extgcd(b,a%b,d,y,x); y=-x*(a/b); }
5 }
6 ll inverse(ll a,ll n){
7     ll d,x,y;
8     extgcd(a,n,d,x,y);
9     return d==1?(x+n)%n:-1;
10 }
```

## 5 计算几何

### 5.1 点初始模板

```

1 #include <cstdio>
2 #include <cmath>
3
4 using namespace std;
5
6 const double pi = acos(-1.0);
7 const double inf = 1e100;
8 const double eps = 1e-6;
9 int sgn(double d)
10 {
11     if(fabs(d) < eps) //fabs返回d绝对值
12         return 0;
13     if(d > 0)
14         return 1;
15     return -1;
16 }
17 int dcmp(double x, double y) //判断浮点数是否相等
18 {
19     if(fabs(x - y) < eps)
20         return 0;
21     if(x > y)
22         return 1;
23     return -1;
24 }
25 int main() {
26     double x = 1.49999;
27     int fx = floor(x); //向下取整函数
28     int cx = ceil(x); //向上取整函数
29     int rx = round(x); //四舍五入函数
30     printf("%f %d %d %d\n", x, fx, cx, rx);
31     //输出结果 1.499990 1 2 1
32     return 0 ;
33 }
```

### 5.2 一些基本运算

```

1 Vector operator + (Vector A, Vector B) //加
2 {
3     return Vector(A.x+B.x, A.y+B.y);
4 }
```

```

1 Vector operator - (Point A, Point B) //减
2 {
3     return Vector(A.x-B.x, A.y-B.y);
4 }
```



```

1 Vector operator * (Vector A, double p) //乘
2 {
3     return Vector(A.x*p, A.y*p);
4 }

```

```

1 Vector operator / (Vector A, double p) //除
2 {
3     return Vector(A.x/p, A.y/p);
4 }

```

```

1 bool operator < (const Point& a, const Point& b) //点
   集升序排列
2 {
3     if(a.x == b.x)
4         return a.y < b.y;
5     return a.x < b.x;
6 }

```

```

1 bool operator == (const Point& a, const Point& b) //
   等于运算
2 {
3     if(dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0)
4         return true;
5     return false;
6 }

```

```

1 double Dot(Vector A, Vector B) //点乘
2 {
3     return A.x*B.x + A.y*B.y;
4 }

```

```

1 double Cross(Vector A, Vector B) //叉乘
2 {
3     return A.x*B.y-A.y*B.x;
4 }

```

```

1 double Length(Vector A) //取模
2 {
3     return sqrt(Dot(A, A));
4 }

```

```

1 double Angle(Vector A, Vector B) //两向量夹角 弧度制
2 {
3     return acos(Dot(A, B) / Length(A) / Length(B));
4 }

```

```

1 double Area2(Point A, Point B, Point C) //俩向量构成平
   行四边形有向面积
2 {
3     return Cross(B-A, C-A);
4 }

```

```

1 Vector Rotate(Vector A, double rad) //向量逆时针旋转后
   的向量
2 { //rad为弧度 且为逆时针旋转的角
3     return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(
4         rad)+A.y*cos(rad));
5 }

```

```

1 Vector Normal(Vector A) //向量逆时针旋转九十度的单位法向
   量
2 { //向量A左转90°的单位法向量
3     double L = Length(A);
4     return Vector(-A.y/L, A.x/L);
5 }

```

```

1 bool ToLeftTest(Point a, Point b, Point c) //判断折线
   bc是不是向ab的逆时针方向(左边)转向
2 {
3     return Cross(b - a, c - b) > 0;
4 }

```

### 5.3 点模板总结

```

1 struct Point{
2     double x, y;
3     Point(double x = 0, double y = 0):x(x),y(y){}
4 };
5 typedef Point Vector;
6 Vector operator + (Vector A, Vector B){
7     return Vector(A.x+B.x, A.y+B.y);
8 }
9 Vector operator - (Point A, Point B){
10    return Vector(A.x-B.x, A.y-B.y);
11 }
12 Vector operator * (Vector A, double p){
13    return Vector(A.x*p, A.y*p);
14 }
15 Vector operator / (Vector A, double p){
16    return Vector(A.x/p, A.y/p);
17 }
18 bool operator < (const Point& a, const Point& b){
19    if(a.x == b.x)
20        return a.y < b.y;
21    return a.x < b.x;
22 }
23 const double eps = 1e-6;
24 int sgn(double x){
25    if(fabs(x) < eps)
26        return 0;
27    if(x < 0)
28        return -1;
29    return 1;
30 }
31 bool operator == (const Point& a, const Point& b){
32    if(sgn(a.x-b.x) == 0 && sgn(a.y-b.y) == 0)
33        return true;
34    return false;
35 }
36 double Dot(Vector A, Vector B){
37    return A.x*B.x + A.y*B.y;
38 }
39 double Length(Vector A){
40    return sqrt(Dot(A, A));
41 }
42 double Angle(Vector A, Vector B){
43    return acos(Dot(A, B)/Length(A)/Length(B));
44 }
45 double Cross(Vector A, Vector B){
46    return A.x*B.y-A.y*B.x;
47 }
48 double Area2(Point A, Point B, Point C){

```



```

49     return Cross(B-A, C-A);
50 }
51 Vector Rotate(Vector A, double rad){//rad为弧度 且为逆
    时针旋转的角
52     return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(
        rad)+A.y*cos(rad));
53 }
54 Vector Normal(Vector A){//向量A左转90°的单位法向量
55     double L = Length(A);
56     return Vector(-A.y/L, A.x/L);
57 }
58 bool ToLeftTest(Point a, Point b, Point c){
59     return Cross(b - a, c - b) > 0;
60 }

```

## 5.4 线

```

1 struct Line
2 { //直线定义
3     Point v, p;
4     Line(Point v, Point p):v(v), p(p) {}
5     Point point(double t){ //返回点 P = v + (p - v)*t
6         return v + (p - v)*t;
7     }
8 };

```

```

1 //计算两直线交点
2 //调用前需保证 Cross(v, w) != 0
3 Point GetLineIntersection(Point P, Vector v, Point Q,
    Vector w)
4 {
5     Vector u = P-Q;
6     double t = Cross(w, u)/Cross(v, w);
7     return P+v*t;
8 }

```

```

1 //点P到直线AB距离公式
2 double DistanceToLine(Point P, Point A, Point B)
3 {
4     Vector v1 = B-A, v2 = P-A;
5     return fabs(Cross(v1, v2)/Length(v1));
6 } //不去绝对值, 得到的是有向距离

```

```

1 //点P到线段AB距离公式
2 double DistanceToSegment(Point P, Point A, Point B){
3     if(A == B)
4         return Length(P-A);
5     Vector v1 = B-A, v2 = P-A, v3 = P-B;
6     if(dcmp(Dot(v1, v2)) < 0)
7         return Length(v2);
8     if(dcmp(Dot(v1, v3)) > 0)
9         return Length(v3);
10    return DistanceToLine(P, A, B);
11 }

```

```

1 //点P在直线AB上的投影点
2 Point GetLineProjection(Point P, Point A, Point B)
3 {
4     Vector v = B-A;
5     return A+v*(Dot(v, P-A)/Dot(v, v));
6 }

```

```

1 //点是否在线段上
2 bool OnSegment(Point p, Point a1, Point a2)
3 {
4     return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1
        -p, a2-p)) <= 0;
5 }

```

```

1 //两线段是否相交
2 //不允许交点在顶点处
3 bool SegmentProperIntersection(Point a1, Point a2,
    Point b1, Point b2){
4     double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2
        - a1, b2 - a1);
5     double c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2
        - b1, a2 - b1);
6     return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) <
        0);
7 }
8 //允许在端点处相交
9 bool SegmentProperIntersection(Point a1, Point a2,
    Point b1, Point b2){
10    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1,
        b2-a1);
11    double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1,
        a2-b1);
12    //if判断控制是否允许线段在端点处相交, 根据需要添加
13    if(!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4))
14    {
15        bool f1 = OnSegment(b1, a1, a2);
16        bool f2 = OnSegment(b2, a1, a2);
17        bool f3 = OnSegment(a1, b1, b2);
18        bool f4 = OnSegment(a2, b1, b2);
19        bool f = (f1|f2|f3|f4);
20        return f;
21    }
22    return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) <
        0);
23 }

```

## 6 字符串

### 6.1 函数

```

1 标准库
2 sscanf
3 sscanf(const char *__source, const char *__format,
    ...) : 从字符串 __source 里读取变量, 比如 sscanf(
        str, "%d", &a) 。
4
5 sprintf
6 sprintf(char *__stream, const char *__format, ...) :
    将 __format 字符串里的内容输出到 __stream 中, 比如
    sprintf(str, "%d", i) 。
7
8 strcmp
9 int strcmp(const char *str1, const char *str2): 按照字
    典序比较 str1 str2 若 str1 字典序小返回负值, 一样返
    回 0, 大返回正值 请注意, 不要简单的认为只有 0, 1, -1
    三种, 在不同平台下的返回值都遵循正负, 但并非都是 0,
    1, -1
10
11 strcpy

```

```

12 char *strcpy(char *str, const char *src) : 把 src 中
    的字符复制到 str 中, str src 均为字符数组头指针, 返
    回值为 str 包含空终止符号 '\0' 。
13
14 strncpy
15 char *strncpy(char *str, const char *src, int cnt) :
    复制至多 cnt 个字符到 str 中, 若 src 终止而数量未达
    cnt 则写入空字符到 str 直至写入总共 cnt 个字符。
16
17 strcat
18 char *strcat(char *str1, const char *str2) : 将 str2
    接到 str1 的结尾, 用 *str2 替换 str1 末尾的 '\0' 返
    回 str1 。
19
20 strstr
21 char *strstr(char *str1, const char *str2) : 若 str2
    是 str1 的子串, 则返回 str2 在 str1 的首次出现的地
    址; 如果 str2 不是 str1 的子串, 则返回 NULL 。
22
23 strchr
24 char *strchr(const char *str, int c) : 找到在字符串
    str 中第一次出现字符 c 的位置, 并返回这个位置的地址。
    如果未找到该字符则返回 NULL 。
25
26 strrchr
27 char *rchr(const char *str, char c) : 找到在字符串
    str 中最后一次出现字符 c 的位置, 并返回这个位置的地址。
    如果未找到该字符则返回 NULL 。

```

## 6.2 Manacher

```

1 const int N = 11000000 + 5;
2 char s[N], ma[N*2];
3 int mp[N*2]; //mp[i] 为在s中以对应位置为中心的极大子回文
    串的总长度+1
4 int Manacher(char *s, int n){
5     int len = 0;
6     ma[len++] = '$'; ma[len++] = '#';
7     for(int i=0;i<n;i++){
8         ma[len++] = s[i];
9         ma[len++] = '#';
10    }
11    ma[len] = 0;
12    int mx = 0, id = 0;
13    int res = 0;
14    for(int i=0;i<len;i++){
15        mp[i] = mx > i ? min(mp[2*id-i], mx - i) : 1;
16        while(ma[i+mp[i]] == ma[i-mp[i]]) mp[i]++;
17        if(i + mp[i] > mx) {
18            mx = mp[i] + i;
19            id = i;
20        }
21        res = max(res, mp[i] - 1);
22    }
23    return res;
24 }
25 int main(){
26     scanf("%s", s);
27     printf("%d", Manacher(s, strlen(s)));
28     return 0;
29 }

```

## 6.3 KMP

```

1 //以 1 为起点, 求nxt数组
2 nxt[1] = 0;
3 for(int i=2,j=0; i<=n; i++){
4     while(j > 0 && a[i] != a[j+1]) j = nxt[j];
5     if(a[i] == a[j+1]) j++;
6     nxt[i] = j;
7 }
8
9 //求 f 匹配数组
10 for(int i=1,j=0;i<=m;i++){
11     while(j > 0 && (j == n || b[i] != a[j+1])) j=nxt[j];
12     if(b[i] == a[j+1]) j++;
13     f[i] = j;
14     //if(f[i] == n) 此时为A在B中的某一次出现
15 }

```

## 7 背包

### 7.1 01

```

1 for(int i = 1; i <= n; ++i)
2 {
3     for(int j = m; j >= 0; --j)
4         if(j >= w[i])
5             f[j] = max(f[j], f[j-w[i]] + v[i]);
6 }

```

### 7.2 完全背包

```

1 cin >> n >> m;
2 for(int i = 1; i <= n; i++) cin >> v[i] >> w[i];
3 f[0] = 0;
4 for(int i = 1; i <= n; i++){
5     for(int j = v[i]; j <= m; j++){
6         f[j] = max(f[j], f[j-v[i]] + w[i]);
7         // cout << f[j] << endl;
8     }
9 }
10 for(int i = 0; i <= m; i++)
11     ans = max(ans, f[i]);
12 cout << ans << endl;

```

### 7.3 多重背包

有  $N$  种物品和一个容量是  $V$  的背包。

第  $i$  种物品最多有  $s_i$  件, 每件体积是  $v_i$ , 价值是  $w_i$ 。

求解将哪些物品装入背包, 可使物品体积总和不超过背包容量, 且价值总和最大。输出最大价值。

```

1 cin>>n>>m;
2 while(n--){
3     {
4         cin>>v>>w>>s;
5         while(s--){
6             {a[++t]=v;
7             b[t]=w;} //死拆, 把多重背包拆成01背包
8         }
9     }
10 }

```

```

9   for(int i=1;i<=t;i++)
10  for(int j=m;j>=a[i];j--)
11  dp[j]=max(dp[j-a[i]]+b[i],dp[j]); //直接套01背包的板子

```

有  $N$  种物品和一个容量是  $V$  的背包。

第  $i$  种物品最多有  $s_i$  件，每件体积是  $v_i$ ，价值是  $w_i$ 。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。输出最大价值。

```

1   int N,V,v[1001],w[1001],dp[2001],s[1001]
2   int a[25000],b[25000]; //2的12次方大于2000，也就是
   说一个数最多可以拆成12个，故数组容量乘12
3   cin>>N>>V;
4   memset(dp,0,sizeof(dp));
5   for(int i=0;i<N;i++)
6   cin>>v[i]>>w[i]>>s[i];
7   int total=0;
8   for(int i=0;i<N;i++)
9   {
10      for(int j=1;j<=s[i];j<=1) //二进制拆分
11      {
12          a[total]=j*w[i]; //存价值
13          b[total++]=j*v[i]; //存容量
14          s[i]-=j;
15      }
16      if(s[i]>0) //当s[i]>0;
17      {
18          a[total]=s[i]*w[i];
19          b[total++]=s[i]*v[i];
20      }
21  }
22  for(int i=0;i<total;i++) //01背包
23  for(int j=V;j>=b[i];j--)
24  dp[j]=max(dp[j],dp[j-b[i]]+a[i]);
25  cout<<dp[V];
26  return 0;
27 }

```

## 7.4 分组背包

有  $N$  组物品和一个容量是  $V$  的背包。

每组物品有若干个，同一组内的物品最多只能选一个。每件物品的体积是  $v_{ij}$ ，价值是  $w_{ij}$ ，其中  $i$  是组号， $j$  是组内编号。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

```

1   for(int i=0;i<n;i++){
2       cin>>s[i];
3       for(int j=0;j<=s[i];j++){
4           cin>>v[i][j]>>w[i][j];
5       }
6   }
7
8   for(int i=0;i<n;i++){
9       for(int j=V;j>=0;j--){
10          for(int k=0;k<=s[i];k++){ //for(int k=s[i];k
   >=1;k--)也可以
11              if(j>=v[i][k]) f[j]=max(f[j],f[j-v[i][k]]+w[i][k]);
12          }
13      }
14  }

```